NPS ARCHIVE
1997.12
JOHNSON, B.

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

THE MIE SCATTERING SERIES AND CONVERGENCE ACCELERATION

by

Brian E. Johnson

December, 1997

Thesis Advisor: James Luscombe
Second Reader: D. Scott Davis

Thesis
J5715

Approved for public release; distribution is unlimited.

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1997	Master's Thesis

4. TITLE AND SUBTITLE
THE MIE SCATTERING SERIES AND CONVERGENCE ACCELERATION

5. FUNDING NUMBERS

6. AUTHOR(S)
Johnson, Brian E.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

In this thesis we present an algorithm for the precise determination of the Mie extinction efficiency parameter. The mathematical representation of the Mie parameters is in the form of an infinite series, and any technique that could be found to accelerate the convergence of the Mie series would have great commercial and military application. Results are presented that show the comparison of the rate of convergence obtained by directly summing the individual terms of the extinction efficiency parameter and the rate obtained using an existing series acceleration technique. It was found that the acceleration method we employed, known as the Levin method of series transformation, proved unsuccessful in accelerating the convergence of the Mie series. However, other acceleration techniques exist and should be explored.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Mie scattering, Levin method, series acceleration	76
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Approved for public release; distribution is unlimited

THE MIE SCATTERING SERIES AND CONVERGENCE ACCELERATION

Brian E. Johnson
Lieutenant, United States Navy
B.S., University of California at Davis, 1991

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN APPLIED PHYSICS

from the

NAVAL POSTGRADUATE SCHOOL
December, 1997

NPS Archive
1997.12
Johnson, B.

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOO
MONTEREY CA 93943-5101

ABSTRACT

In this thesis we present an algorithm for the precise determination of the Mie extinction efficiency parameter. The mathematical representation of the Mie parameters is in the form of an infinite series, and any technique that could be found to accelerate the convergence of the Mie series would have great commercial and military application. Results are presented that show the comparison of the rate of convergence obtained by directly summing the individual terms of the extinction efficiency parameter and the rate obtained using an existing series acceleration technique. It was found that the acceleration method we employed, known as the Levin method of series transformation, proved unsuccessful in accelerating the convergence of the Mie series. However, other acceleration techniques exist and should be explored.

TABLE OF CONTENTS

I. INTRODUCTION..... 1
II. THE MIE SCATTERING PARAMETERS..... 7
A. BACKGROUND..... 7
B. SCATTERING AMPLITUDE PARAMETERS..... 8
C. SCATTERING EFFICIENCY AND EXTINCTION PARAMETERS..... 8
D. THE MIE COEFFICIENTS..... 9
E. CALCULATING Q_{ext} USING ANSI C..... 11
III. THE LEVIN METHOD..... 15
A. SERIES ACCELERATION METHODS..... 15
B. THE E-ALGORITHM..... 17
C. LEVIN'S TRANSFORMS..... 18
D. ENCODING THE LEVIN TRANSFORM..... 20
IV. RESULTS..... 23
V. CONCLUSION..... 25
APPENDIX A. ALTERNATING HARMONIC SERIES CODE IN ANSI C..... 27
APPENDIX B. OUTPUT OF ALTERNATING HARMONIC SERIES CODE..... 29
APPENDIX C. LEVIN ALGORITHM ON ALTERNATING HARMONIC SERIES..... 31
APPENDIX D. ANSI C CODE TO PRODUCE DATA FOR ALTERNATING HARMONIC SERIES PLOT..... 33
APPENDIX E. ANSI C CODE TO PRODUCE DATA FOR PLOT OF LEVIN METHOD ON ALTERNATING HARMONIC SERIES..... 35
APPENDIX F. MATLAB CODE TO PRODUCE PLOT OF CONVERGENCE OF ALTERNATING HARMONIC SERIES..... 37
APPENDIX G. ANSI C CODE TO PRODUCE DATA FOR EXTINCTION EFFICIENCY FACTOR AND CONVERGENCE RATE VS SIZE PARAMETER..... 39
APPENDIX H. MATLAB CODE TO PLOT OUTPUT OF CODE LISTED IN APPENDIX G..... 47
APPENDIX I. ANSI C CODE FOR CALCULATING EXTINCTION EFFICIENCY FACTOR..... 49
APPENDIX J. ANSI C CODE FOR LEVIN ALGORITHM OPERATING ON EXTINCTION EFFICIENCY FACTOR..... 57
LIST OF REFERENCES..... 65
INITIAL DISTRIBUTION LIST..... 67

I. INTRODUCTION

Mie scattering is the scattering of electromagnetic radiation by primarily spherical particles whose diameters are comparable to the wavelength of the incident radiation [Ref. 1: p. 303]. In nature, we see this manifested as the white appearance of clouds. Due to the distribution of particle sizes in the clouds, the cumulative effect of the Mie scattered light waves on the water droplets is that the whole spectrum of scattered radiation combines to make the cloud appear white. In other scenarios which involve the direct use of modern technology, Mie scattering has many applications where it is of vital importance to have a highly efficient and reliable algorithm for signal processing and for analyzing the properties of Mie scattered waves. A few examples include the reflection of a radar signal off of a cloud of dispersed particles and the absorption and scattering of radiation from soot particles in the atmosphere [Ref. 2]; Mie scattering as a technique for the sizing of air bubbles [Ref. 3]; and for remote sensing applications. Further applications can be found in fields as diverse as astrophysics [Refs. 4 - 5], physical chemistry [Refs. 6-7], and a unique style of painting that employs the Mie scattering phenomenon [Ref. 8].

In order to determine the amplitude, extinction efficiency, or scattering efficiency of an electromagnetic wave that has undergone Mie scattering, it is necessary to compute the mathematical representation of these quantities, or Mie parameters, each of which is analytically represented by an infinite sum. For the purpose of this thesis research, our attention has been focused on calculating the Mie parameter known as the extinction

efficiency factor, or Q_{ext}, which describes the total effect of scattering and absorption in removing radiation from the incident beam.

As will be shown later in Chapter IV, the number of terms required for the infinite series to converge numerically to a given accuracy is directly proportional to the ratio between the wavelength of the incident radiation and the circumference of the scattering particle. The thrust of this thesis research has been to determine if this infinite series can be caused to converge in dramatically fewer iterations through the utilization of a series acceleration technique. The basic idea is to transform the Mie series into another series that converges faster. While numerous series acceleration methods are known, one of the most powerful is known as the Levin method [Ref. 9: p. 35]. The ultimate goal of this thesis research has been to determine if the Levin method can significantly accelerate the convergence of the Mie series.

As an example of the power of the Levin method (detailed in chapter 3), consider the alternating harmonic series that represents the natural logarithm of 2, namely,

$$\ln(2) = \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n} = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \cdots \quad . \tag{1}$$

The convergence of this series is extremely slow. By summing the individual terms of Eq. (1), over a million and a half terms must be taken to achieve an accuracy of six digits: 1,565,239 terms to be exact. The C code used to determine this value and the program output are included as Appendices A and B, respectively. By employing the Levin method, however, the same degree of numerical accuracy can be reached after only six iterations! The ANSI C code for the Levin method operating on the alternating harmonic

series is included as Appendix C. After the fifteenth term of the Levin method is reached, the accepted value of accuracy out to the sixteenth digit is obtained [Ref. 10: p. 113]. This is an astounding reduction of computer processing time and is an amazing demonstration of the power of the Levin method of series acceleration. (We note that, to obtain eight digit accuracy by straightforward summation of the series would require over one hundred million terms; nine digit accuracy was effectively unattainable using the UNIX system resources available for this research.) Table 1 shows a comparison of the partial sum obtained from Eq. (1) and those calculated with the Levin method. As a further comparison, the plot shown in Fig. 1 was produced to show the relative speed (i.e., number of iterations required) with which the Levin method and the brute force method of summing the individual terms of the alternating harmonic series converge to 99

$$\ln 2 = 0.6931471805599453$$

Iterations	Partial Sum	Levin Result
1	1.000000	1.0000000000000000
2	0.500000	0.6666666666666666
3	0.833333	0.6944444444444444
4	0.583333	0.6931372549019608
5	0.783333	0.6931439393939393
6	0.616667	0.6931474019283136
7	0.759524	0.6931471777900349
8	0.634524	0.6931471800150043
9	0.745635	0.6931471806012293
10	0.645635	0.6931471805592415
11	0.736544	0.6931471805598542
12	0.653211	0.6931471805599532
13	0.730134	0.6931471805599452
14	0.658705	0.6931471805599454
15	0.725372	0.6931471805599453

Table 1. Levin Method on Logarithm of 2.

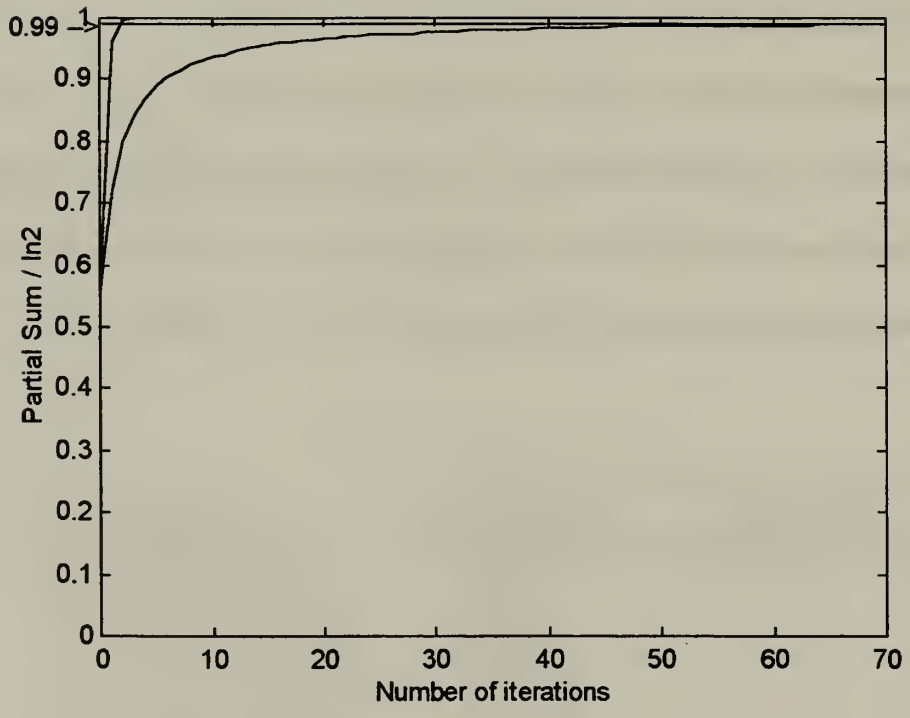

Figure 1. Levin Method vs. Infinite Series for ln(2)

percent of the correct value of ln(2). Appendices D through F show the pertinent ANSI C and Matlab code. Note that the Levin method required only three terms to reach 99 percent of the value of ln(2), while summing the individual terms of the series required sixty-five terms. As a final comparison, the time required to reach six digits of accuracy using the brute force method was determined (Appendices A and B), and the code used in Appendix C was altered to determine the number of times the Levin method could reach the same degree of accuracy in the same time span (11.7 seconds). The value obtained was approximately 20,000, indicating that in this instance the Levin method is about 20,000 times faster than summing the individual terms of the infinite series.

In light of the preceding example, the question arises as to whether the Levin method can accelerate the Mie series. A computer program was written to calculate the individual terms of the infinite series representation of the Mie parameter Q_{ext} and to determine the number of these terms required for the series to converge to an accuracy of one part in 10^8. Next, the Levin method was applied to this infinite series, and the number of iterations of the Levin method required to achieve a similar accuracy was likewise determined. A comparison of these two results showed that the Levin method did *not* succeed in accelerating the convergence of the Mie series. In fact, the number of iterations required for the Levin method to converge was comparable to that obtained by simply summing the infinite series representation of the Mie parameter. Considering the algorithm which the Levin method utilizes, this method when applied to the Mie series will actually result in *increased* processor time. Although the Levin method proved

unsuccessful in accelerating the convergence of the Mie series, further research in this area is still needed as there are other series acceleration methods worth exploring [Ref. 11: p. 56].

In the next section, we discuss the relevant Mie parameters and the infinite series for computing the extinction coefficient. In Chapter III we discuss the Levin convergence acceleration method. In Chapter IV we discuss our results for applying the Levin method to the Mie series.

II. THE MIE SCATTERING PARAMETERS

A. BACKGROUND

There are four basic Mie scattering parameters. The first two pertain to the scattering amplitude and describe the complex amplitudes of the perpendicular and parallel components (with respect to the scattering plane) of the electric field vector. The third Mie parameter is the scattering efficiency factor, while the fourth parameter is the extinction efficiency factor. All four of these expressions are given in dimensionless terms, and the evaluation of these four basic functions relies on the proper evaluation of four coefficients: the Mie coefficients a_n and b_n, and the angular coefficients π_n and τ_n.

These four expressions can be described using three basic parameters: a dimensionless size x of the scattering sphere, an index of refraction m (usually complex), and a scattering angle θ (relative to the forward direction). The dimensionless size parameter x is given by

$$x = kr = \left(\frac{2\pi}{\lambda}\right) r = \frac{circumference\ of\ sphere}{wavelength} \quad , \tag{2}$$

where $k = 2\pi/\lambda$ is the wave number, r is the radius of the sphere, and λ is the wavelength of the incident radiation. The complex index of refraction m is given by

$$m = \upsilon - \kappa i \quad , \tag{3}$$

where ν is the real part of the index of refraction and κ is the imaginary part. A complex refractive index indicates an absorbing sphere, such as soot. A purely real index of

refraction is indicative of a non-absorbing sphere; for the purpose of this thesis research, only cases of purely real refractive index were considered.

B. SCATTERING AMPLITUDE PARAMETERS

Let the electric field vector amplitude of the scattered radiation field be given by $\mathbf{A}_{SC}$. This radiation field can in turn be expressed in terms of the scalar perpendicular and parallel components A_1 and A_2. We can define dimensionless, complex amplitudes S_1 and S_2 by multiplying the amplitudes A_1 and A_2 by the free-space propagation constant k, which in turn can be represented by an infinite converging series [Ref. 12: p. 13]:

$$kA_1 \equiv S_1(m,x,\theta) = \sum_{n=1}^{\infty} \frac{2n+1}{n(n+1)} \left\{ a_n \pi_n(\mu) + b_n \tau_n(\mu) \right\} \quad , \tag{4a}$$

$$kA_2 \equiv S_2(m,x,\theta) = \sum_{n=1}^{\infty} \frac{2n+1}{n(n+1)} \left\{ b_n \pi_n(\mu) + a_n \tau_n(\mu) \right\} \quad . \tag{4b}$$

The Mie coefficients a_n and b_n are functions of the index of refraction m and the size x, while the angular coefficients π_n and τ_n are functions of μ=cosθ only. The latter coefficients are defined in terms of Legendre polynomials and their derivatives. The coefficients a_n and b_n are functions of spherical Bessel functions of the first, second, and third kind, and can be expressed in a variety of ways.

C. SCATTERING EFFICIENCY AND EXTINCTION PARAMETERS

It can be shown that the differential scattering cross section for unit incident flux is

$$d\sigma(m,x,\theta) = \tfrac{1}{2} A_{SC} \cdot A_{SC}^{*}(m,x,\theta) d\omega \quad , \tag{5}$$

where $d\omega$ is an element of the solid angle.

Following Diermendjian [Ref. 12: p. 13], by representing the unpolarized incident radiation as the sum of two independent and linearly polarized components of equal flux, the scattering cross section can be expressed as

$$\sigma_{sca}(m,x) = \int_{\Omega} d\sigma(m,x,\theta) = \frac{1}{2}\int_{\Omega}(A_1 A_1^* + A_2 A_2^*)d\omega \quad , \tag{6}$$

where $\Omega = 4\pi$ is the solid angle.

The scattering efficiency factor $Q_{SC}(m,x)$ is obtained by normalizing σ_{sca} by the sphere's geometrical cross section πr^2:

$$Q_{sca}(m,x) = \frac{\sigma_{sca}(m,x)}{\pi r^2} = \frac{2}{x^2}\sum_{n=1}^{\infty}(2n+1)(|a_n|^2 + |b_n|^2) \quad . \tag{7}$$

The total extinction cross section and efficiency factor, which includes the contribution due to absorption, can be similarly defined. This leads to the following expression for extinction efficiency factor:

$$Q_{ext} = \frac{2}{x^2}\sum_{n=1}^{N}(2n+1)\,\mathrm{Re}(a_n + b_n) \quad . \tag{8}$$

As noted earlier for the case of a purely real index of refraction m, there will be no absorption, and thus the extinction efficiency factor Q_{ext} will be identically equal to the scattering efficiency factor Q_{sca} .

D. THE MIE COEFFICIENTS

The simplest and most elegant expressions for the Mie coefficients are as follows [Ref. 13: p. 195]:

$$a_n = \frac{j_n(kr)}{h^{(1)}{}_n(kr)} \quad , \tag{9a}$$

$$b_n = \frac{krj_n(kr) - nj_n(kr)}{krh_{n-1}(kr) - nh^{(1)}{}_n(kr)} \quad , \tag{9b}$$

where $j_n(kr)$ is a spherical Bessel function of the first kind with order n, $h^{(1)}{}_n(kr)$ is a Bessel function of the third kind, known as a Hankel function, and is defined as

$$h^{(1)}{}_n(kr) = j_n(kr) + iy_n(kr) \quad , \tag{10}$$

and $y_n(kr)$ is a spherical Bessel function of the second kind with order n.

Another common form of the Mie coefficients is that adopted by van de Hulst [Ref. 14: p. 123]:

$$a_n = \frac{A_n(z)\psi_n(x) - m\psi'_n(x)}{A_n(z)\zeta_n(x) - m\zeta'_n(x)} \quad , \tag{11a}$$

$$b_n = \frac{mA_n(z)\psi_n(x) - \psi'_n(x)}{mA_n(z)\zeta_n(x) - \zeta'_n(x)} \quad , \tag{11b}$$

where $z = mx$, while ψ_n and ζ_n are Ricatti-Bessel functions which are defined as follows:

$$\psi_n(x) = xj_n(x) \quad , \tag{12a}$$

$$\chi_n(x) = -xy_n(x) \quad , \tag{12b}$$

$$\zeta_n(x) = \psi_n(x) + i\chi_n(x) \quad . \tag{12c}$$

Here, j_n and y_n are spherical Bessel functions of the first and second kind, respectively.

For computational purposes, it is best to express the Mie coefficients in a form conducive to separating their real and imaginary components. Through the use of recursion formulas and circular functions [Ref. 12: p. 16-19], the Mie coefficients may be written in the alternative form [Ref. 15: pp. 16-17]:

$$a_n = \frac{\left\{\frac{A_n(z)}{m} + \frac{n}{x}\right\}\psi_n(x) - \psi_{n-1}(x)}{\left\{\frac{A_n(z)}{m} + \frac{n}{x}\right\}\zeta_n(x) - \zeta_{n-1}(x)}, \tag{13a}$$

$$b_n = \frac{\left\{mA_n(z) + \frac{n}{x}\right\}\psi_n(x) - \psi_{n-1}(x)}{\left\{mA_n(z) + \frac{n}{x}\right\}\zeta_n(x) - \zeta_{n-1}(x)}. \tag{13b}$$

These forms of the Mie coefficients were programmed, using the C language, to determine the Mie coefficients and hence the extinction efficiency factor Q_{ext}.

E. CALCULATING Q_{ext} USING ANSI C

The author wishes to thank the writers of "Numerical Recipes in C" [Ref. 16] for enabling the circumvention of the reinvention of the wheel, in that their code for computing ordinary and spherical Bessel functions and their derivatives was utilized in this thesis research for computing the Mie coefficients and calculating the extinction efficiency factor. Of noteworthy interest is that said code cites Jerry Lentz of the Naval Postgraduate School as the author of an improved technique for calculating Bessel functions through the use of continued fractions.

The code used for calculating the ordinary Bessel functions is incorporated into a structure which returns the values of the Bessel functions of the first and second kind, and their first derivatives. The spherical Bessel function code simply generates a normalization factor, makes a call to the function which calculates ordinary Bessel functions (of half-odd integer order), then in like fashion returns values of the spherical Bessel functions of the first and second kind, and their first derivatives. The code for calculating the ordinary

Bessel functions also makes calls to two additional functions, "beschb" and "chebev", also obtained from "Numerical Recipes in C" [Ref. 16]. The "beschb" function is used to evaluate the gamma functions present in the Bessel function formula, calling the "chebev" function in the process to perform Chebyshev expansion.

Once the interfacing codes for calculating ordinary and spherical Bessel functions were brought together, a program was written to assemble the components of the Mie coefficients as expressed in Eqs. (13a) and (13b). Again, the Mie coefficients are functions of the dimensionless size parameter x and the index of refraction m.

Next, a loop was created that calculated the individual terms of the extinction efficiency factor (one of the Mie parameters), each term of which relied on a calculated value of the Mie coefficients. As is the case of all of the Mie parameters, the extinction efficiency factor is represented as an infinite sum. Thus, the loop for calculating the efficiency extinction factor was summed until the value converged to one part in 10^8. With each increment of the loop counter, the Bessel functions associated with the Mie coefficients were of subsequent higher order, i.e., for n=1, $J_n(x)$ would be a Bessel function of the first order; for n=2, $J_n(x)$ would be a Bessel function of the second order, and so on.

As a test for the accuracy of the code and the correctness of the programming, a routine was created that calculated the extinction efficiency factor for size parameter x varying from 0 to 9, using a step size of 0.01 and applying 6 different values for the refractive index. A plot showing all six curves was produced from the data obtained from this program and is shown in Fig. 2. This plot very closely resembles that which was

produced by Van de Hulst [Ref. 14: p. 151], with the exception that Van de Hulst's plot was intentionally smoother as a result of not showing fine detail. The C code used to produce data for Fig. 2 is included as Appendix G, with the output data being sent to a file called "Qout". The Matlab code used to produce Figure 2 is included as Appendix H.

III. THE LEVIN METHOD

A. SERIES ACCELERATION METHODS

The main idea behind the Levin method, as in any series acceleration technique, is to transform a given slowly converging series into another series that converges faster. We note here the distinction between a sequence and a series. An infinite sequence (S_n) is an unending progression of numbers S_n which may be real or complex. A sequence converges if a number, S, exists so that, corresponding to every positive number ε, no matter how small, a number n_o can be found such that $|S_n - S| < \varepsilon$ for $n > n_o$. In this case, the sequence (S_n) is said to converge to the limit S as n tends to infinity. An infinite series, on the other hand, is the sum of an infinite sequence. Let $u_1, u_2, \ldots, u_n, \ldots$ be an infinite sequence of numbers, real or complex. Let the sum $u_1 + u_2 + \cdots u_n$ be denoted by S_n, which is called the n^{th} partial sum. Then, if the sequence of partial sums (S_n) converges to a limit, S, the infinite series $u_1 + u_2 + \cdots$ is said to be convergent, or to converge to the sum S. The connection with the Levin method is to transform the sequence of partial sums into another sequence that converges faster, i.e., requires fewer terms to converge to the limit.

Given a sequence of real or complex numbers (S_n) which converges to S, this sequence can be transformed into another sequence (T_n). A trivial example of such a sequence transformation is

$$T_n = \frac{S_n + S_{n+1}}{2}, \qquad n = 0,1,\ldots . \tag{14}$$

To be useful, however, the following properties for the transformed sequence must hold [Ref. 11: p. 1]:

1. (T_n) must converge,
2. (T_n) must converge to the same limit as (S_n), and
3. (T_n) must converge to S faster than (S_n); that is, $\lim_{n\to\infty}(T_n - S)/(S_n - S) = 0$.

If property 3 holds, the transformation T is said to *accelerate the convergence* of the sequence (S_n), or that the sequence (T_n) *converges faster* than (S_n). An example of a sequence transformation meeting these three conditions is Aitken's Δ^2 process [Ref. 11: pp. 1-7]:

$$T_n = \frac{\Delta S_{n+1}}{\Delta^2 S_n} \cdot S_n + \left(1 - \frac{\Delta S_{n+1}}{\Delta^2 S_n}\right) \cdot S_{n+1} \quad , \tag{15}$$

or,

$$T_n = \frac{S_n S_{n+2} - S_{n+1}^2}{S_{n+2} - 2S_{n+1} + S_n} \quad , \qquad n = 0,1,\ldots \ . \tag{16}$$

where Δ is the difference operator defined by $\Delta v_n = v_{n+1} - v_n$ and $\Delta^{k+1} v_n = \Delta^k v_{n+1} - \Delta^k v_n$. The Δ^2 in the denominator of Eq. (15) accounts for the process's name.

The Levin transformation, as with most other transformation algorithms, is a particular case of what is known as the E-transformation, which is the most general sequence transformation. Transformations belonging to the E-transformation class include [Ref. 11: p. 56]: Richardson polynomial extrapolation, Shanks' transformation, the Levin transformation, and many other historically known acceleration methods. We note that

series acceleration is an active field of research in numerical analysis, with the Levin method and E-transformation having been discovered only in the past 25 years.

B. THE E-ALGORITHM

Following Brezinski and Zaglia [Ref. 11: pp. 56-57], the E-transformation is based on the following relation:

$$S_n - S - a_1 g_1(n) - \cdots - a_k g_k(n) = 0 \quad , \tag{17}$$

where S_n is an element of the sequence to be transformed, the a's are unknown scalars, the $g_i(n)$'s are given auxiliary sequences (which may depend on terms of the sequence S_n itself), and where k is a fixed integer. Rewriting Eq. (17) in terms of S_n,

$$S_n = S + a_1 g_1(n) + \cdots + a_k g_k(n) \quad . \tag{18}$$

The basic idea behind Eq. (18) is to attempt to fit the actual behavior of S_n, as a function of n, so that it may be extrapolated smoothly to the (unknown) limit, S. This fitting is achieved through the particular choices of the functions $g_i(n)$. By incrementing n in Eq. (18) to $n + k$, one has $k + 1$ equations in $k + 1$ unknowns (the limit S and the scalars $a_1, \cdots, a_k$). By solving these equations, we obtain a sequence of estimates for the limit, which we denote by $E_k^{(n)}$:

$$E_k^{(n)} = \frac{\begin{vmatrix} S_n & \cdots & S_{n+k} \\ g_1(n) & \cdots & g_1(n+k) \\ \vdots & & \vdots \\ g_k(n) & \cdots & g_k(n+k) \end{vmatrix}}{\begin{vmatrix} 1 & \cdots & 1 \\ g_1(n) & \cdots & g_1(n+k) \\ \vdots & & \vdots \\ g_k(n) & \cdots & g_k(n+k) \end{vmatrix}} \quad . \tag{19}$$

It is assumed that the determinant in the denominator of Eq. (19) is not equal to zero. Finally, note that Eq. (18) is the kernel of the transformation Eq. (19); that is, Eq. (18) is the set of sequences for which there exists the sequence S such that, for all n, $E_{k,}^{(n)} = S$.

The E-algorithm is the recursive algorithm that allows one to compute the numbers $E_k^{(n)}$ without actually computing the determinants in Eq. (19). Given the following rules,

$$E_0^{(n)} = S_n \ , \qquad n = 0, 1, \ldots \tag{20a}$$

$$g_{0,i}^{(n)} = g_i(n) \ , \qquad n = 0, 1, \ldots \text{ and } i = 1, 2, \ldots \ , \tag{20b}$$

then for $k = 1, 2, \ldots$ and $n = 0, 1, \ldots$, the main rule for the E-algorithm is

$$E_k^{(n)} = E_{k-1}^{(n)} - \frac{E_{k-1}^{(n+1)} - E_{k-1}^{(n)}}{g_{k-1,k}^{(n+1)} - g_{k-1,k}^{(n)}} \cdot g_{k-1,k}^{(n)} \ , \tag{21}$$

where the $g_{k-1,k}^{(n)}$'s are auxiliary sequences computed by the following auxiliary rule:

$$g_{k,i}^{(n)} = g_{k-1,i}^{(n)} - \frac{g_{k-1,i}^{(n+1)} - g_{k-1,i}^{(n)}}{g_{k-1,k}^{(n+1)} - g_{k-1,k}^{(n)}} \cdot g_{k-1,k}^{(n)} \ , \qquad i = k+1, k+2, \ldots \ . \tag{22}$$

C. LEVIN'S TRANSFORMS

Levin's method of generating non-linear transformations for increasing the rate of convergence of sequences [Ref. 17: pp. 371-388] was first introduced in 1973. For Levin's generalized transformation, the auxiliary sequences denoted by the $g_i(n)$'s in the E-transformation given in Eq. (19) take on the particular form:

$$g_i(n) = \frac{x_n^{i-1} \Delta S_n}{y_n} \ , \tag{23}$$

where x_n and y_n are themselves auxiliary sequences.

Levin's transforms are basically generalizations of Aitken's Δ^2 process and of the E-transformation corresponding to the first column of the E-algorithm as shown in Eq. (21) [Ref. 11: p. 113], denoted as E_1. The kernel of Aitken's process is

$$S_n - S = a \cdot \Delta S_n \quad , \tag{24}$$

while the kernel of the transformation E_1 is of the form

$$S_n - S = t_n \cdot g(n) \quad . \tag{25}$$

By definition in the Levin method, $g(n)$ is taken to be an arbitrary polynomial of degree $(k-1)$ of the quantity $(n)^{-1}$. Also, t_n denotes the n^{th} element of the series (as for example the n^{th} term of the Mie series). Thus, the sequences of Eq. (25) can be expressed as

$$S_n - S = t_n \cdot \left[a_1 + a_2 (n)^{-1} + \cdots + a_k (n)^{-(k-1)} \right] \quad . \tag{26}$$

Following Brezinski and Zaglia, multiplying both sides of Eq. (26) by $(n)^{k-1}$ and rearranging terms yields

$$(n)^{(k-1)} \cdot \frac{S_n - S}{t_n} = a_1 (n)^{(k-1)} + a_2 (n)^{k-2} + \cdots + a_k \quad . \tag{27}$$

By applying the operator Δ^k to both sides of Eq. (27), the right hand side becomes identically zero, since it is a polynomial of degree $(k-1)$ in n. Therefore, for all n,

$$\Delta^k \left((n)^{(k-1)} \cdot \frac{S_n - S}{t_n} \right) = 0 \quad . \tag{28}$$

Since Δ^k is a linear operator, the above expression can be also be expressed as

$$\Delta^k \left((n)^{(k-1)} \cdot \frac{S_n}{t_n} \right) = S \cdot \Delta^k \left(\frac{(n)^{(k-1)}}{t_n} \right) \quad . \tag{29}$$

The numbers $E_k^{(n)}$, denoted now as the Levin estimate $L_N^{(n)}$, are therefore given by

$$L_N^{(n)} = \frac{\Delta^N\left((n)^{(N-1)} \cdot S_n / t_n\right)}{\Delta^N\left((n)^{(N-1)} / t_n\right)} \quad . \tag{30}$$

Using the well-known formula [Ref. 11: p. 115]

$$\Delta^N u_n = \sum_{k=0}^{N} (-1)^k C_N^k u_{n+k} \quad , \tag{31}$$

where C_N^k is the binomial coefficient defined by

$$\binom{N}{k} = \frac{N!}{k!(N-k)!} \quad , \tag{32}$$

Eq. (30) then becomes

$$L_N^{(n)} = \frac{\sum_{k=1}^{N} (-1)^k (n+k)^{N-1} \frac{S_{n+k}}{t_{n+k}} N! \Big/ k!(N-k)!}{\sum_{k=1}^{N} (-1)^k (n+k)^{N-1} \frac{1}{t_{n+k}} N! \Big/ k!(N-k)!} \quad . \tag{33}$$

Equation (33) defines the sequence of Levin estimates for the limit of the series.

D. ENCODING THE LEVIN TRANSFORM

In the instances where the Levin code (Appendices C and L) was utilized, T_N was used to represent the partial sum S_{n+k} from Eq. (33) above. The sum T_N is related to the sequence t_k by the following expression:

$$T_N = \sum_{k=1}^{N} t_k \quad . \tag{34}$$

In other words, t_k represents the individual term for the loop which calculates the extinction efficiency factor Q_{ext}, and T_N is the value for which Q_{ext} converges to one part

in 10^8. Equation (33) was split into two individually calculated terms, $P[N]$ and $Q[N]$, representing the numerator and denominator, respectively. Consequently, the following equations result:

$$P[N]=\sum_{k=1}^{N}(-1)^k(k)^{N-1}\frac{T_k}{t_k}N!\Big/k!(N-k)!\ , \text{ and} \tag{35}$$

$$Q[N]=\sum_{k=1}^{N}(-1)^k(k)^{N-1}\frac{1}{t_k}N!\Big/k!(N-k)!\ . \tag{36}$$

Thus, Eq. (30) becomes

$$L_N^{(n)}=T[N]=\frac{P[N]}{Q[N]}\ . \tag{37}$$

Equation (34) was encoded in order to calculate the Levin estimate of Q_{ext}, and the number of terms needed for the Levin transform to cause the infinite series representation of Q_{ext} to converge was subsequently determined. This number was compared with that which was obtained by direct summing of the individual terms of the infinite series representation of Q_{ext} to convergence in order to determine if the Levin method accelerated the convergence.

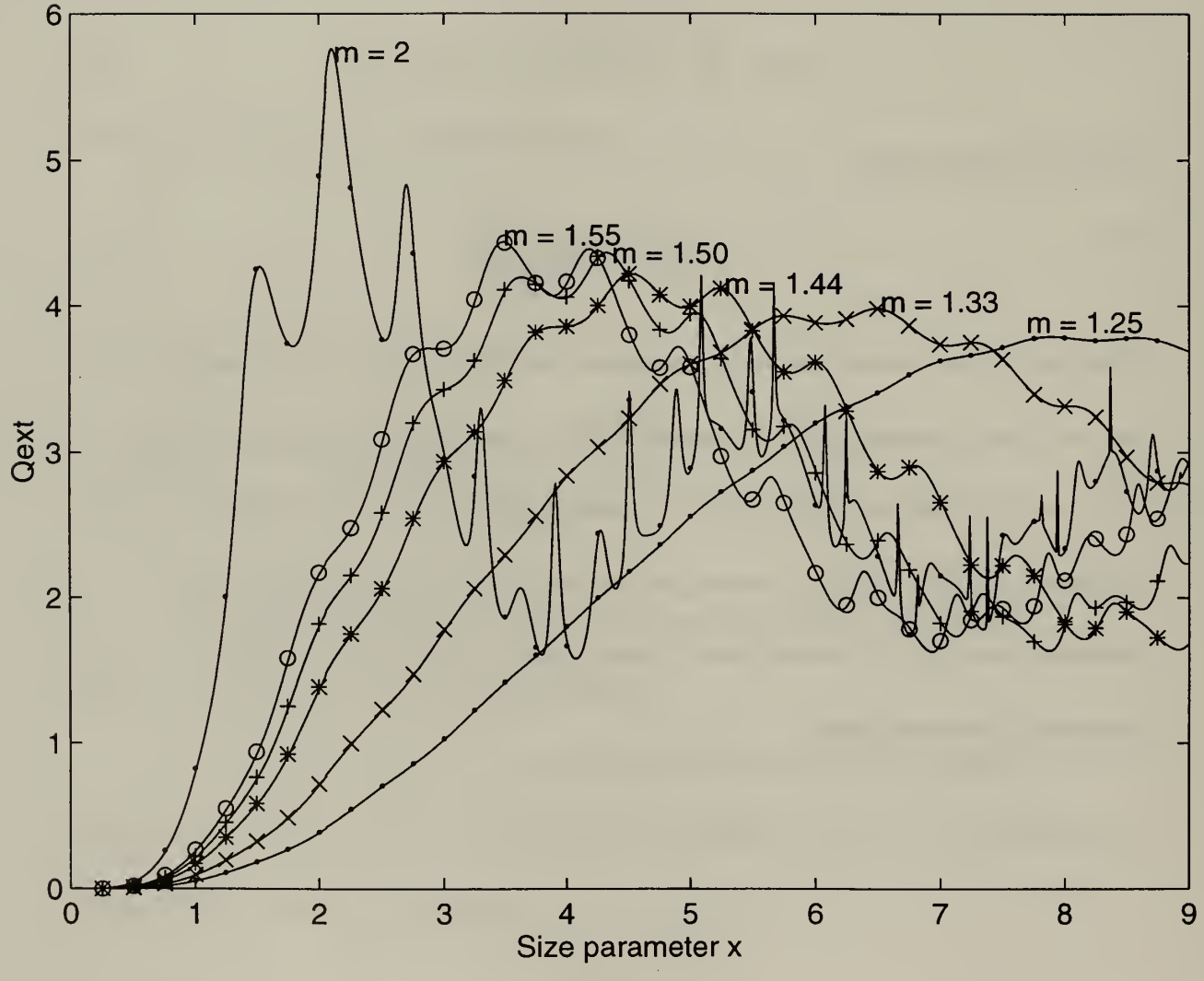

Figure 2. Extinction Efficiency Factor vs. Size Parameter

IV. RESULTS

After completing the Mie code and producing values for the extinction efficiency factor that agreed with the tabulated results of Van de Hulst, the result obtained from applying the Levin method of series acceleration to the infinite series representation of a Mie scattering parameter was somewhat anti-climactic, albeit not entirely unanticipated. Simply put, the Levin method did not accelerate the convergence of the extinction efficiency factor. In point of fact, the number of iterations required for the Levin method to converge to a given accuracy was approximately the same, less one or two terms, as the number of terms required by direct summation. Size parameters ranging from 0 to 100 were used in making this determination. Since the Levin method requires calls to additional functions that calculate the log of a gamma function, the log of a factorial, and a binomial coefficient at each iteration of the summation loop – the codes for which were obtained from "Numerical Recipes in C" [Ref. 16] – this method is actually slower than direct summation. It should also be noted that, for size parameters $x > 108$, the Levin method fails entirely because the individual terms t_n become vanishingly small, producing undefined output. The code used to determine Q_{ext} via summing the infinite series and the code utilizing the Levin estimate are included as Appendices I and J, respectively.

The C code used to produce the data used for the plot shown in Figure 2 was slightly altered to determine the number of iterations required for the Mie series to converge as a function of size parameter. The result is shown in Figure 3. Since both methods of convergence required the same number of iterations, the code used for

summing the individual terms of the Mie series (Appendix G) was chosen to generate the data used for the plot shown in Figure 3. A linear relationship can easily be seen for all indices of refraction, with the number of terms in the series proportional to the size parameter.

While a tremendous amount of effort was put forth to come to the conclusion that the Levin method is unsuccessful in accelerating the convergence of the Mie series, the work was not all for naught. An important question was answered, leading the way for further research in this area.

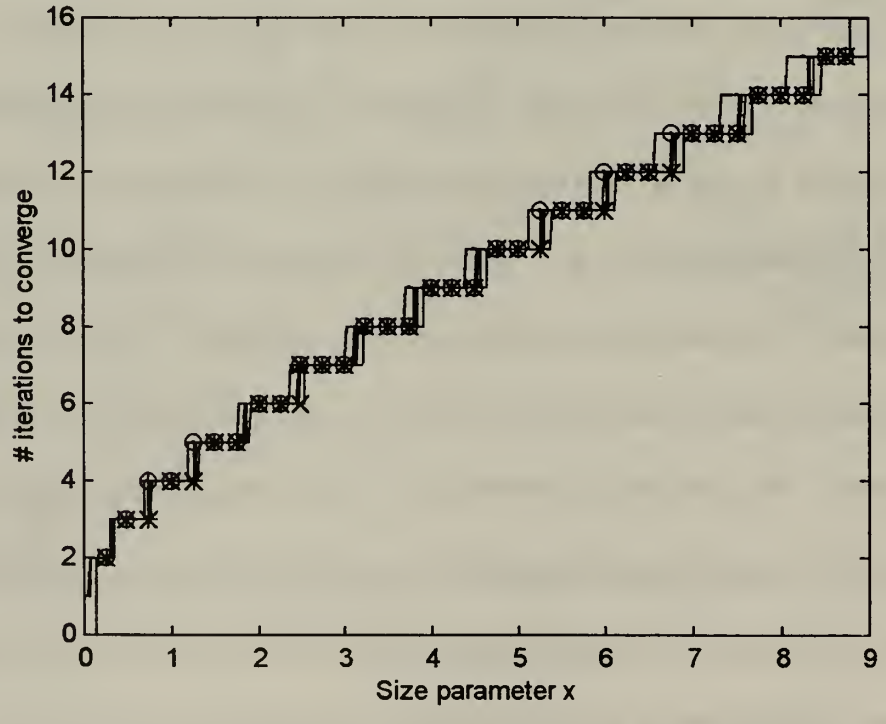

Figure 3. Rate Convergence vs. Size Parameter

V. CONCLUSION

In this thesis research we assembled the code to calculate the Mie coefficients, integral components of all four of the Mie scattering parameters. Next, we calculated the Mie extinction efficiency factor by summing the terms of the infinite series representation to convergence. Several different indices of refraction were used, and a plot of the extinction efficiency factor versus size parameter was produced. These results were found to agree closely with that of Van de Hulst's authoritative work, thereby ensuring the accuracy of the code. Finally, an algorithm for the Levin method of series transformation was incorporated into the existing code. The rate of convergence was subsequently determined and compared with the rate of convergence achieved by summing the individual terms of the Mie series. Based on the results, we arrived at several important conclusions.

First, the number of terms required to converge was found to be linearly related to the size parameter. This included size parameters both within and outside the Mie regime. The linear relationship continued until a limitation was reached, which leads us to our second conclusion.

The magnitude of the size parameter was found to impose a limit on the extent to which the Levin method was able to cause the Mie series to converge. While a size parameter of 108 is well beyond the Mie scattering region and within the optical region, it still points to a computational limit of the Levin method.

Most importantly, we have concluded that the Levin method of series transformation did not accelerate the convergence of the Mie series. Several simulations were run using size parameters ranging from less than one up to one hundred, and indices of refraction ranging from one to two. All results showed a rate of convergence, insofar as the number of iterations required, equal to the rate obtained by summing the individual terms of the Mie series. This is an important result. There are other series transformation methods in existence that may accelerate the convergence of the Mie series, and these should be explored in their own right.

APPENDIX A: ALTERNATING HARMONIC SERIES CODE IN ANSI C

```
/* Thesis Program to calculate number of terms for alternating harmonic series to reach progressive */
/* digits of accuracy */
/* LT Brian Johnson */
/* Compiler:  Borland C++ Ver. 5.0 */
/* File Name:  ln2.c */

# include <stdio.h>
# include <math.h>
# include <time.h>

int n;
int places = 1;

double AltHarmSum = 0.0;
double RoundedNew = 0.0;
double RoundedOld = 0.0;

float accuracy = 10.0;

main()
{
 n = 0;

 do {
  n += 1;
  AltHarmSum+=(double)( pow(-1,n+1)/n);
  RoundedNew = floor(AltHarmSum*accuracy+.5)/accuracy;

  if ( RoundedNew == RoundedOld )
  {
   printf("\nIt took %d terms to reach %d digits of accuracy:", n, places);
   printf("\n  Sum = %f ", AltHarmSum);
   printf("\n  Time elapsed: %3.2f sec.\n", clock()/1e6 );
   places+=1;
   accuracy*=10.0;
  }
  RoundedOld = RoundedNew;
 } while (places <= 15);
}
```

APPENDIX B. OUTPUT OF ALTERNATING HARMONIC SERIES CODE

It took 12 terms to reach 1 digits of accuracy:
Sum = 0.653211
Time elapsed: 0.00 sec.

It took 271 terms to reach 2 digits of accuracy:
Sum = 0.694989
Time elapsed: 0.00 sec.

It took 1417 terms to reach 3 digits of accuracy:
Sum = 0.693500
Time elapsed: 0.01 sec.

It took 177341 terms to reach 4 digits of accuracy:
Sum = 0.693150
Time elapsed: 1.33 sec.

It took 229300 terms to reach 5 digits of accuracy:
Sum = 0.693145
Time elapsed: 1.71 sec.

It took 1565239 terms to reach 6 digits of accuracy:
Sum = 0.693147
Time elapsed: 11.70 sec.

APPENDIX C: LEVIN ALGORITHM ON ALTERNATING HARMONIC SERIES

```
/* Thesis Program to apply Levin method to alternating harmonic series and compute number of terms */
/* for series to converge */
/* LT Brian Johnson */
/* Compiler: Borland C++ Ver. 5.0 */
/* File Name:  ln2lev.c */

#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <math.h>

#define MAX 15

int k;

double T[MAX+1], t[MAX+1];

double a;

main()
{
 T[0] = t[0] = 0.0;
 k = 1;

 printf("\n\ni        T[i]        t[i]        P[i]/Q[i]\n");

 while(k<=MAX)
 {
  t[k] = (double)( pow(-1,k+1)/k);
  T[k] = T[k-1] + t[k];
  k = k + 1;
 }
 a = levin();
}

double levin()
{
 double P[MAX+1],Q[MAX+1];
 int l;
 l=1;
 P[0] = Q[0] = 0.0;
 while(l<=MAX)
 {
  P[l]=P[l-1]+pow(-1,l)*pow(l,MAX-1)*T[l]/t[l]*bico(MAX,l);
  Q[l]=Q[l-1]+pow(-1,l)*pow(l,MAX-1)/t[l]*bico(MAX,l);
  l=l+1;
  printf("\n%d        %f        %f        %f ", i, T[i], t[i], P[i]/Q[i]);
 }
 return (P[MAX]/Q[MAX]);
}
```

```
double bico(int n, int k)
{       double factln(int n);
        return floor(0.5+exp(factln(n)-factln(k)-factln(n-k)));
}

double factln(int n)
{       double gammln(float xx);
        static float a[101];
        if (n <= 1) return 0.0;
        if (n <= 100) return a[n] ? a[n] : (a[n]=gammln(n+1.0));
        else return gammln(n+1.0);
}

double gammln(float xx)
{       double x,y,tmp,ser;
        static double cof[6]={76.18009172947146,-86.50532032941677,
                24.01409824083091,-1.231739572450155,
                0.1208650973866179e-2,-0.5395239384953e-5};
        int j;
        y=x=xx;
        tmp=x+5.5;
        tmp -= (x+0.5)*log(tmp);
        ser=1.000000000190015;
        for (j=0;j<=5;j++) ser += cof[j]/++y;
        return -tmp+log(2.5066282746310005*ser/x);
                                                }
```

APPENDIX D: ANSI C CODE TO PRODUCE DATA FOR ALTERNATING HARMONIC SERIES PLOT

```
/* Thesis Program to produce data for plot of convergence of alternating harmonic series to ln(2) */
/* LT Brian Johnson */
/* Compiler:  Borland C++ Ver. 5.0 */
/* File Name:  ln2plot.c */

# include <stdio.h>
# include <math.h>
# include <time.h>

int n;
int places = 1;

double AltHarmSum = 0.0;
double RoundedNew = 0.0;
double RoundedOld = 0.0;

float accuracy = 10.0;

main()
{
 for (n = 1; n <= 51; n++) {
    AltHarmSum+=(double)( pow(-1,n+1)/n);
    RoundedNew = floor(AltHarmSum*accuracy+.5)/accuracy;

 printf("\n%d  %1.3f", n, log(2) - fabs(log(2) - AltHarmSum));
  }
}
```

APPENDIX E. ANSI C CODE TO PRODUCE DATA FOR PLOT OF LEVIN METHOD ON ALTERNATING HARMONIC SERIES

```
/* Thesis Program to produce data for plot of Levin method on the convergence of alternating */
/* harmonic series to ln(2) */
/* LT Brian Johnson */
/* Compiler:  Borland C++ Ver. 5.0 */
/* File Name:  ln2levplot.c */

#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <math.h>
#include "nrutil.h"
#include "nrutil.c"
#include <time.h>

#define MAX 16

double levin();
double bico(int, int);
double factln(int);
double gammln(float);
double T[MAX+1], t[MAX+1];
double a;

int k, i;

main()
{
 T[0] = t[0] = 0.0;
 k = 1;

 while(k<=MAX)
 {
  t[k] = (double)( pow(-1,k+1)/k);
  T[k] = T[k-1] + t[k];
  k = k + 1;
 }  /* end while */

 a = levin();

}  /* end main */

double levin()
{
 double P[MAX+1], Q[MAX+1];
 double commonTerm;
 int N;
 P[0] = Q[0] = 0.0;

 for (N = 1; N <= MAX; N++)
```

```
  {
   for (k = 1; k <= N; k++)
   {
    commonTerm = pow(-1,k)*pow(k,N-1)/t[k]*bico(N,k);
    P[k] = P[k-1] + commonTerm*T[k];
    Q[k] = Q[k-1] + commonTerm;
   }

   printf("\n%d     %32.31lf ", k-1, log(2) - fabs(log(2) - P[N]/Q[N]));
  } /* end for N loop */

  printf("\n\n");
  return (P[MAX]/Q[MAX]);
}

double bico(int n, int k)
{       double factln(int n);
        return floor(0.5+exp(factln(n)-factln(k)-factln(n-k)));
}

double factln(int n)
{       double gammln(float xx);
        static float a[101];
        if (n <= 1) return 0.0;
        if (n <= 100) return a[n] ? a[n] : (a[n]=gammln(n+1.0));
        else return gammln(n+1.0);
}

double gammln(float xx)
{       double x,y,tmp,ser;
        static double cof[6]={76.18009172947146,-86.50532032941677,
                24.01409824083091,-1.231739572450155,
                0.1208650973866179e-2,-0.5395239384953e-5};
        int j;
        y=x=xx;
        tmp=x+5.5;
        tmp -= (x+0.5)*log(tmp);
        ser=1.000000000190015;
        for (j=0;j<=5;j++) ser += cof[j]/++y;
        return -tmp+log(2.5066282746310005*ser/x);
                                                }
```

APPENDIX F. MATLAB CODE TO PRODUCE PLOT OF CONVERGENCE OF ALTERNATING HARMONIC SERIES

```
/* Thesis Program to produce data for plot of Levin method on the convergence of alternating */
/* harmonic series to ln(2) */
/* LT Brian Johnson */
/* Software:  Matlab Ver. 4.2C */
/* File Name:  ln2comp.m */

load ln2graf -ascii
load ln2levgr -ascii
x = ln2graf(:,1);
y = ln2graf(:,2);

x2 = ln2levgr(:,1);
y2 = ln2levgr(:,2);

B = ones(1,71);
figure(1)
C = 0.99*B;
plot(x-1,y/log(2),x-1,C,':', x2-1,y2/log(2))
xlabel('Number of iterations')
ylabel('Partial Sum / ln2')
title ('Levin Method vs Infinite Sum for ln(2)')
axis([0 70 0 1])
gtext('0.99 -->')
```

APPENDIX G. ANSI C CODE TO PRODUCE DATA FOR EXTINCTION EFFICIENCY FACTOR AND CONVERGENCE RATE VS SIZE PARAMETER

```
/* Thesis Program to produce data for plotting Q_ext vs size parameter */
/* LT Brian Johnson */
/* Compiler:  Borland C++ Ver. 5.0 */
/* File Name:  Miegraph.c */

#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
/* #include <iostream.h> */
#include <math.h>
#include "nrutil.h"
#include "nrutil.c"
#define NRANSI
#define EPS 1.0e-10
#define FPMIN 1.0e-30
#define MAXIT 10000
#define XMIN 2.0
#define PI 3.141592653589793
#define RTPIO2 1.2533141
#define NUSE1 5
#define NUSE2 5

struct bessVec {
  double rj, ry, rjp, ryp;
};

struct complex {
  double real, imag;
};

void assignBessels(struct bessVec* besselsPtr, double rjdummy, double rydummy, double rjpdummy,
double rypdummy)
{
  besselsPtr -> rj = rjdummy;
  besselsPtr -> ry = rydummy;
  besselsPtr -> rjp = rjpdummy;
  besselsPtr -> ryp = rypdummy;
}

struct bessVec sphbes(double, int);          /* Function Prototype */
struct bessVec bessjy(double, double);       /* Function Prototype */

struct bessVec sphBessels_n;                 /* j's, y's & deriv's for Ricatti-Bes */
struct bessVec bessels_nPlusHalf;            /* for calculating J_n +/- half */
struct bessVec bessels_nMinusHalf;
struct complex a_n, b_n;                     /* Mie coefficients */
struct complex zeta_n;                       /* Ricatti-Bessel fn */
struct complex zeta_nMinus1;
struct complex denom;                        /* denominator in Mie coefficients */
```

```
double x;                          /* size parameter */
double m;                          /* refractive index (non-absorbing spheres) */
double J_nMinusHalf, J_nPlusHalf;  /* for calculating A_factor */
double A_factor;                   /* psi-prime divided by psi */
double psi_nMinus1;                /* Ricatti-Bessel fn 1st kind, initialized */
double psi_n;
double chi_n;                      /* Ricatti-Bessel fn 2nd kind */
double nOverx;
double aBraces, bBraces;
double numer;                      /* numerator in Mie coefficients */
double realDenom;                  /* rationalized denominator */
double sum_QextOld, sum_Qext;      /* Extinction efficiency factor */
double Q_ext[900][6];              /* array of Q_ext's, fn's of m and x */
int num[900][6];                   /* number of iterations to converge, f(x,m) */
double conv = 1.0e-8;

int i;                             /* counter for x loop */
int j;                             /* counter for mArray loop */
int n;                             /* # iterations to converge, also order of bessel function */

main()
{
double mArray[6] = {1.25, 1.33, 1.44, 1.50, 1.55, 2.0};



for(j = 0; j <= 5; j++) {

  m = mArray[j];

  for(x = 0.01; x <=9.0; x += 0.01) {

    n = 0;                                    /* initialize n for each x */
    bessels_nPlusHalf = bessjy(m*x, 0.5);     /* initialize for A_factor */
    J_nPlusHalf = bessels_nPlusHalf.rj;

    psi_n = sin(x);
    zeta_n.real = psi_n;                      /* initializing zeta */
    zeta_n.imag = cos(x);

    sum_Qext =  0.0;

    do {
      n = n + 1;
      J_nMinusHalf = J_nPlusHalf;
      psi_nMinus1 = psi_n;
      zeta_nMinus1 = zeta_n;
      sum_QextOld = sum_Qext;

      bessels_nPlusHalf = bessjy(m*x, n + 0.5);     /* calculating A factor */
      J_nPlusHalf = bessels_nPlusHalf.rj;
      A_factor = (J_nMinusHalf/J_nPlusHalf) - (n/(m*x));
```

```
        sphBessels_n = sphbes(x, n);            /* j's and y's for Ricatti Bess fn's */
        psi_n = x * sphBessels_n.rj;
        chi_n = -x * sphBessels_n.ry;
        zeta_n.real = psi_n;
        zeta_n.imag = chi_n;

        nOverx = n/x;
        aBraces = (A_factor / m + nOverx);
        bBraces = (A_factor * m + nOverx);

        numer = aBraces*psi_n - psi_nMinus1;

        denom.real = (aBraces * zeta_n.real) - zeta_nMinus1.real;
        denom.imag = (aBraces * zeta_n.imag) - zeta_nMinus1.imag;
        realDenom = (denom.real*denom.real + denom.imag*denom.imag);

        a_n.real = (numer * denom.real)/realDenom;
        a_n.imag = (numer * denom.imag)/realDenom;

        numer = bBraces*psi_n - psi_nMinus1;

        denom.real = (bBraces * zeta_n.real) - zeta_nMinus1.real;
        denom.imag = (bBraces * zeta_n.imag) - zeta_nMinus1.imag;
        realDenom = (denom.real*denom.real + denom.imag*denom.imag);

        b_n.real = (numer * denom.real)/realDenom;
        b_n.imag = (numer * denom.imag)/realDenom;

        sum_Qext += (2*n + 1)*(a_n.real + b_n.real);

      } while (fabs(sum_Qext - sum_QextOld) > conv);     /* end do-while loop */

      Q_ext[(int)(x*100-1)][j] = 2.0/(x*x)*sum_Qext;
      num[(int)(x*100-1)][j] = n;

    }                   /* end inner x-loop */
}                       /* end outer j-loop (for various m values) */

for (i = 0; i < 899; i++) {
   printf("\n%f  ", (i+1.0)/100.0);

   for (j = 0; j <= 5; j++)
      printf("%f  ", Q_ext[i][j]);
/*       printf("%d  ", num[i][j]); */

} /* end i-j loop */


  return 0;
}

struct bessVec sphbes(double x, int n)
{
```

```
        struct bessVec tempsphbessels;
        struct bessVec bessels;

        double sj, sy, sjp, syp;
        void nrerror(char error_text[]);
        double factor, order;

        if (n < 0 || x <= 0.0) nrerror("bad arguments in sphbes");
        order=n+0.5;

        bessels = bessjy(x,order);
        factor=RTPIO2/sqrt(x);
        sj=factor*bessels.rj;
        sy=factor*bessels.ry;
        sjp=factor*bessels.rjp-(sj)/(2.0*x);
        syp=factor*bessels.ryp-(sy)/(2.0*x);

        assignBessels(&tempsphbessels, sj, sy, sjp, syp);

        return tempsphbessels;

}

struct bessVec bessjy(double x, double xnu)
{
        void nrerror(char error_text[]);

        struct bessVec tempBessels;

        double rj, ry, rjp, ryp;

        void beschb(double x, double *gam1, double *gam2, double *gampl,
                double *gammi);
        int i,isign,l,nl;
        double a,b,br,bi,c,cr,ci,d,del,del1,den,di,dlr,dli,dr,e,f,fact,fact2,
                fact3,ff,gam,gam1,gam2,gammi,gampl,h,p,pimu,pimu2,q,r,rjl,
                rjl1,rjmu,rjp1,rjpl,rjtemp,ry1,rymu,rymup,rytemp,sum,sum1,
                temp,w,x2,xi,xi2,xmu,xmu2;


        if (x <= 0.0 || xnu < 0.0) nrerror("bad arguments in bessjy");
        nl=(x < XMIN ? (int)(xnu+0.5) : IMAX(0,(int)(xnu-x+1.5)));
        xmu=xnu-nl;
        xmu2=xmu*xmu;
        xi=1.0/x;
        xi2=2.0*xi;
        w=xi2/PI;
        isign=1;
        h=xnu*xi;
        if (h < FPMIN) h=FPMIN;
        b=xi2*xnu;
        d=0.0;
        c=h;
```

```
for (i=1;i<=MAXIT;i++) {
        b += xi2;
        d=b-d;
        if (fabs(d) < FPMIN) d=FPMIN;
        c=b-1.0/c;
        if (fabs(c) < FPMIN) c=FPMIN;
        d=1.0/d;
        del=c*d;
        h=del*h;
        if (d < 0.0) isign = -isign;
        if (fabs(del-1.0) < EPS) break;
}
if (i > MAXIT) nrerror("x too large in bessjy; try asymptotic expansion");
rjl=isign*FPMIN;
rjpl=h*rjl;
rjl1=rjl;
rjp1=rjpl;
fact=xnu*xi;
for (l=nl;l>=1;l--) {
        rjtemp=fact*rjl+rjpl;
        fact -= xi;
        rjpl=fact*rjtemp-rjl;
        rjl=rjtemp;
}
if (rjl == 0.0) rjl=EPS;
f=rjpl/rjl;
if (x < XMIN) {
        x2=0.5*x;
        pimu=PI*xmu;
        fact = (fabs(pimu) < EPS ? 1.0 : pimu/sin(pimu));
        d = -log(x2);
        e=xmu*d;
        fact2 = (fabs(e) < EPS ? 1.0 : sinh(e)/e);
        beschb(xmu,&gam1,&gam2,&gampl,&gammi);
        ff=2.0/PI*fact*(gam1*cosh(e)+gam2*fact2*d);
        e=exp(e);
        p=e/(gampl*PI);
        q=1.0/(e*PI*gammi);
        pimu2=0.5*pimu;
        fact3 = (fabs(pimu2) < EPS ? 1.0 : sin(pimu2)/pimu2);
        r=PI*pimu2*fact3*fact3;
        c=1.0;
        d = -x2*x2;
        sum=ff+r*q;
        sum1=p;
        for (i=1;i<=MAXIT;i++) {
                ff=(i*ff+p+q)/(i*i-xmu2);
                c *= (d/i);
                p /= (i-xmu);
                q /= (i+xmu);
                del=c*(ff+r*q);
                sum += del;
                del1=c*p-i*del;
```

```
                sum1 += del1;
                if (fabs(del) < (1.0+fabs(sum))*EPS) break;
        }
        if (i > MAXIT) nrerror("bessy series failed to converge");
        rymu = -sum;
        ry1 = -sum1*xi2;
        rymup=xmu*xi*rymu-ry1;
        rjmu=w/(rymup-f*rymu);
} else {
        a=0.25-xmu2;
        p = -0.5*xi;
        q=1.0;
        br=2.0*x;
        bi=2.0;
        fact=a*xi/(p*p+q*q);
        cr=br+q*fact;
        ci=bi+p*fact;
        den=br*br+bi*bi;
        dr=br/den;
        di = -bi/den;
        dlr=cr*dr-ci*di;
        dli=cr*di+ci*dr;
        temp=p*dlr-q*dli;
        q=p*dli+q*dlr;
        p=temp;
        for (i=2;i<=MAXIT;i++) {
                a += 2*(i-1);
                bi += 2.0;
                dr=a*dr+br;
                di=a*di+bi;
                if (fabs(dr)+fabs(di) < FPMIN) dr=FPMIN;
                fact=a/(cr*cr+ci*ci);
                cr=br+cr*fact;
                ci=bi-ci*fact;
                if (fabs(cr)+fabs(ci) < FPMIN) cr=FPMIN;
                den=dr*dr+di*di;
                dr /= den;
                di /= -den;
                dlr=cr*dr-ci*di;
                dli=cr*di+ci*dr;
                temp=p*dlr-q*dli;
                q=p*dli+q*dlr;
                p=temp;
                if (fabs(dlr-1.0)+fabs(dli) < EPS) break;
        }
        if (i > MAXIT) nrerror("cf2 failed in bessjy");
        gam=(p-f)/q;
        rjmu=sqrt(w/((p-f)*gam+q));
        rjmu=SIGN(rjmu,rjl);
        rymu=rjmu*gam;
        rymup=rymu*(p+q/gam);
        ry1=xmu*xi*rymu-rymup;
}
```

```
        fact=rjmu/rjl;
        rj=rjl1*fact;
        rjp=rjp1*fact;
        for (i=1;i<=nl;i++) {
                rytemp=(xmu+i)*xi2*ry1-rymu;
                rymu=ry1;
                ry1=rytemp;
        }
        ry=rymu;
        ryp=xnu*xi*rymu-ry1;

        assignBessels(&tempBessels, rj, ry, rjp, ryp);

        return tempBessels;
}

void beschb(double x, double *gam1, double *gam2, double *gampl, double *gammi)
{
        double chebev(double a, double b, double c[], int m, double x);
        double xx;
        static double c1[] = {
                -1.142022680371172e0,6.516511267076e-3,
                3.08709017308e-4,-3.470626964e-6,6.943764e-9,
                3.6780e-11,-1.36e-13};
        static double c2[] = {
                1.843740587300906e0,-0.076852840844786e0,
                1.271927136655e-3,-4.971736704e-6,-3.3126120e-8,
                2.42310e-10,-1.70e-13,-1.0e-15};

        xx=8.0*x*x-1.0;
        *gam1=chebev(-1.0,1.0,c1,NUSE1,xx);
        *gam2=chebev(-1.0,1.0,c2,NUSE2,xx);
        *gampl= *gam2-x*(*gam1);
        *gammi= *gam2+x*(*gam1);
}

double chebev(double a, double b, double c[], int m, double x)
{
        void nrerror(char error_text[]);
        double d = 0.0, dd = 0.0, sv, y, y2;
        int j;

        if ((x-a)*(x-b) > 0.0) nrerror("x not in range in routine chebev");
        y2 = 2.0*(y=(2.0*x-a-b)/(b-a));
        for (j = m - 1; j >= 1; j--) {
                sv = d;
                d = y2*d - dd + c[j];
                dd = sv;
        }
        return y*d - dd + 0.5*c[0];
}

#undef EPS
```

```
#undef FPMIN
#undef MAXIT
#undef XMIN
#undef PI
#undef NRANSI
#undef NUSE1
#undef NUSE2
#undef RTPIO2
```

APPENDIX H. MATLAB CODE TO PLOT OUTPUT OF CODE LISTED IN APPENDIX G.

```
load Qout -ascii
x = Qout(:,1);
y1 = Qout(:,2);
y2 = Qout(:,3);
y3 = Qout(:,4);
y4 = Qout(:,5);
y5 = Qout(:,6);
y6 = Qout(:,7);

step = 0.25;
xinc = step:step:8.75;
for i = 1:(length(xinc)-1)
  y1inc(i) = y1(100*xinc(i)+1);
  y2inc(i) = y2(100*xinc(i)+1);
  y3inc(i) = y3(100*xinc(i)+1);
  y4inc(i) = y4(100*xinc(i)+1);
  y5inc(i) = y5(100*xinc(i)+1);
  y6inc(i) = y6(100*xinc(i)+1);
end

plot(x,y1,x,y2,x,y3,x,y4,x,y5,x,y6, xinc,y1inc,'.', xinc,y2inc,'x', xinc, y3inc,'*', xinc,y4inc,'+',
xinc,y5inc,'o', xinc, y6inc,'.')
axis([0,9,0,6])
xlabel('Size parameter x'), ylabel('Qext')
gtext('m = 2')
gtext('m = 1.55')
gtext('m = 1.50')
gtext('m = 1.44')
gtext('m = 1.33')
gtext('m = 1.25')

load Qoutn -ascii
x = Qoutn(:,1);
y1 = Qoutn(:,2);
y2 = Qoutn(:,3);
y3 = Qoutn(:,4);
y4 = Qoutn(:,5);
y5 = Qoutn(:,6);
y6 = Qoutn(:,7);

step = 0.25;
xinc = step:step:8.75;

for i = 1:(length(xinc)-1)
  y1inc(i) = y1(100*xinc(i)+1);
  y2inc(i) = y2(100*xinc(i)+1);
  y3inc(i) = y3(100*xinc(i)+1);
  y4inc(i) = y4(100*xinc(i)+1);
  y5inc(i) = y5(100*xinc(i)+1);
```

```
  y6inc(i) = y6(100*xinc(i)+1);
end

plot(x,y1,'k',x,y2,'m',x,y3,'c',x,y4,'r',x,y5,'g',x,y6,'b', xinc,y1inc,'.k', xinc,y2inc,'xm', xinc, y3inc,'*c',
xinc,y4inc,'+r', xinc,y5inc,'og', xinc, y6inc,'.b')

                    xlabel('Size parameter x'), ylabel('# iterations to converge')
```

APPENDIX I. ANSI C CODE FOR CALCULATING EXTINCTION EFFICIENCY FACTOR

```
/* Thesis Program for calculating Qext as a function of user-defined size parameter */
/* LT Brian Johnson */
/* Compiler: Borland C++ Ver. 5.0 */
/* File Name: mie.c */


#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <math.h>
#include "nrutil.h"
#include "nrutil.c"
#define NRANSI
#define EPS 1.0e-10
#define FPMIN 1.0e-30
#define MAXIT 10000
#define XMIN 2.0
#define PI 3.141592653589793
#define RTPIO2 1.2533141
#define NUSE1 5
#define NUSE2 5

struct bessVec {
  double rj, ry, rjp, ryp;
};

struct complex {
  double real, imag;
};

void assignBessels(struct bessVec* besselsPtr, double rjdummy, double rydummy, double rjpdummy,
double rypdummy)
{
  besselsPtr -> rj = rjdummy;
  besselsPtr -> ry = rydummy;
  besselsPtr -> rjp = rjpdummy;
  besselsPtr -> ryp = rypdummy;
}

struct bessVec sphbes(double, int);                    /* Function Prototype */
struct bessVec bessjy(double, double);                 /* Function Prototype */

struct bessVec sphBessels_n;
struct bessVec bessels_nPlusHalf;
struct bessVec bessels_nMinusHalf;
struct complex a_n, b_n;          /* Mie coefficients */
struct complex zeta_n;
struct complex zeta_nMinus1;
struct complex denom;                  /* denominator in Mie coefficients */
```

```
double x = 109.0;                    /* size parameter */
double m = 2.0;                      /* refractive index (non-absorbing spheres) */
double J_nMinusHalf, J_nPlusHalf;
double A_factor;
double psi_nMinus1;                  /* Ricatti-Bessel fn 1st kind, initialized */
double psi_n;
double chi_n;                        /* Ricatti-Bessel fn 2nd kind */
double nOverx;
double aBraces, bBraces;
double numer;                        /* numerator in Mie coefficients */
double realDenom;                    /* rationalized denominator */
double sum_QextOld, sum_Qext, Q_ext;     /* Extinction efficiency factor */
double conv = 1.0e-8;


int n = 0;                   /* loop counter, also order of bessel function */

main()
{
bessels_nPlusHalf = bessjy(m*x, 0.5);        /* initialize for A_factor */
J_nPlusHalf = bessels_nPlusHalf.rj;

psi_n = sin(x);
zeta_n.real = psi_n;                         /* initializing zeta */
zeta_n.imag = cos(x);

sum_Qext =  0.0;

do {
  n = n + 1;
  J_nMinusHalf = J_nPlusHalf;
  psi_nMinus1 = psi_n;
  zeta_nMinus1 = zeta_n;
  sum_QextOld = sum_Qext;

  bessels_nPlusHalf = bessjy(m*x, n + 0.5);      /* calculating A factor */
  J_nPlusHalf = bessels_nPlusHalf.rj;
  A_factor = (J_nMinusHalf/J_nPlusHalf) - (n/(m*x));

  sphBessels_n = sphbes(x, n);       /* j's and y's for Ricatti Bess fn's */
  psi_n = x * sphBessels_n.rj;
  chi_n = -x * sphBessels_n.ry;
  zeta_n.real = psi_n;
  zeta_n.imag = chi_n;

  nOverx = n/x;
  aBraces = (A_factor / m + nOverx);
  bBraces = (A_factor * m + nOverx);

  numer = aBraces*psi_n - psi_nMinus1;

  denom.real = (aBraces * zeta_n.real) - zeta_nMinus1.real;
```

```
  denom.imag = (aBraces * zeta_n.imag) - zeta_nMinus1.imag;
  realDenom = (denom.real*denom.real + denom.imag*denom.imag);

  a_n.real = (numer * denom.real)/realDenom;
  a_n.imag = (numer * denom.imag)/realDenom;

  numer = bBraces*psi_n - psi_nMinus1;

  denom.real = (bBraces * zeta_n.real) - zeta_nMinus1.real;
  denom.imag = (bBraces * zeta_n.imag) - zeta_nMinus1.imag;
  realDenom = (denom.real*denom.real + denom.imag*denom.imag);

  b_n.real = (numer * denom.real)/realDenom;
  b_n.imag = (numer * denom.imag)/realDenom;

  sum_Qext += (2*n + 1)*(a_n.real + b_n.real);

} while (fabs(sum_Qext - sum_QextOld) > conv);

Q_ext = 2.0/(x*x)*sum_Qext;

printf("\n\nThe Extinction Efficiency Factor is:  Qext = %f", Q_ext, "\n\n");
printf("\nNumber of iterations: %d\n", n);
printf("\nx = %3.2f  m = %1.2f \n\n", x, m);
return 0;
}

struct bessVec sphbes(double x, int n)
{
        struct bessVec tempsphbessels;
        struct bessVec bessels;

        double sj, sy, sjp, syp;
        void nrerror(char error_text[]);
        double factor, order;

        if (n < 0 || x <= 0.0) nrerror("bad arguments in sphbes");
        order=n+0.5;

        bessels = bessjy(x,order);
        factor=RTPIO2/sqrt(x);
        sj=factor*bessels.rj;
        sy=factor*bessels.ry;
        sjp=factor*bessels.rjp-(sj)/(2.0*x);
        syp=factor*bessels.ryp-(sy)/(2.0*x);

        assignBessels(&tempsphbessels, sj, sy, sjp, syp);

        return tempsphbessels;
}

struct bessVec bessjy(double x, double xnu)
{
```

```
void nrerror(char error_text[]);

struct bessVec tempBessels;

double rj, ry, rjp, ryp;

void beschb(double x, double *gam1, double *gam2, double *gampl,
        double *gammi);
int i,isign,l,nl;
double a,b,br,bi,c,cr,ci,d,del,del1,den,di,dlr,dli,dr,e,f,fact,fact2,
        fact3,ff,gam,gam1,gam2,gammi,gampl,h,p,pimu,pimu2,q,r,rjl,
        rjl1,rjmu,rjp1,rjpl,rjtemp,ry1,rymu,rymup,rytemp,sum,sum1,
        temp,w,x2,xi,xi2,xmu,xmu2;


if (x <= 0.0 || xnu < 0.0) nrerror("bad arguments in bessjy");
nl=(x < XMIN ? (int)(xnu+0.5) : IMAX(0,(int)(xnu-x+1.5)));
xmu=xnu-nl;
xmu2=xmu*xmu;
xi=1.0/x;
xi2=2.0*xi;
w=xi2/PI;
isign=1;
h=xnu*xi;
if (h < FPMIN) h=FPMIN;
b=xi2*xnu;
d=0.0;
c=h;
for (i=1;i<=MAXIT;i++) {
        b += xi2;
        d=b-d;
        if (fabs(d) < FPMIN) d=FPMIN;
        c=b-1.0/c;
        if (fabs(c) < FPMIN) c=FPMIN;
        d=1.0/d;
        del=c*d;
        h=del*h;
        if (d < 0.0) isign = -isign;
        if (fabs(del-1.0) < EPS) break;
}
if (i > MAXIT) nrerror("x too large in bessjy; try asymptotic expansion");
rjl=isign*FPMIN;
rjpl=h*rjl;
rjl1=rjl;
rjp1=rjpl;
fact=xnu*xi;
for (l=nl;l>=1;l--) {
        rjtemp=fact*rjl+rjpl;
        fact -= xi;
        rjpl=fact*rjtemp-rjl;
        rjl=rjtemp;
}
if (rjl == 0.0) rjl=EPS;
```

```
f=rjpl/rjl;
if (x < XMIN) {
        x2=0.5*x;
        pimu=PI*xmu;
        fact = (fabs(pimu) < EPS ? 1.0 : pimu/sin(pimu));
        d = -log(x2);
        e=xmu*d;
        fact2 = (fabs(e) < EPS ? 1.0 : sinh(e)/e);
        beschb(xmu,&gam1,&gam2,&gampl,&gammi);
        ff=2.0/PI*fact*(gam1*cosh(e)+gam2*fact2*d);
        e=exp(e);
        p=e/(gampl*PI);
        q=1.0/(e*PI*gammi);
        pimu2=0.5*pimu;
        fact3 = (fabs(pimu2) < EPS ? 1.0 : sin(pimu2)/pimu2);
        r=PI*pimu2*fact3*fact3;
        c=1.0;
        d = -x2*x2;
        sum=ff+r*q;
        sum1=p;
        for (i=1;i<=MAXIT;i++) {
                ff=(i*ff+p+q)/(i*i-xmu2);
                c *= (d/i);
                p /= (i-xmu);
                q /= (i+xmu);
                del=c*(ff+r*q);
                sum += del;
                del1=c*p-i*del;
                sum1 += del1;
                if (fabs(del) < (1.0+fabs(sum))*EPS) break;
        }
        if (i > MAXIT) nrerror("bessy series failed to converge");
        rymu = -sum;
        ry1 = -sum1*xi2;
        rymup=xmu*xi*rymu-ry1;
        rjmu=w/(rymup-f*rymu);
} else {
        a=0.25-xmu2;
        p = -0.5*xi;
        q=1.0;
        br=2.0*x;
        bi=2.0;
        fact=a*xi/(p*p+q*q);
        cr=br+q*fact;
        ci=bi+p*fact;
        den=br*br+bi*bi;
        dr=br/den;
        di = -bi/den;
        dlr=cr*dr-ci*di;
        dli=cr*di+ci*dr;
        temp=p*dlr-q*dli;
        q=p*dli+q*dlr;
        p=temp;
```

```
                for (i=2;i<=MAXIT;i++) {
                        a += 2*(i-1);
                        bi += 2.0;
                        dr=a*dr+br;
                        di=a*di+bi;
                        if (fabs(dr)+fabs(di) < FPMIN) dr=FPMIN;
                        fact=a/(cr*cr+ci*ci);
                        cr=br+cr*fact;
                        ci=bi-ci*fact;
                        if (fabs(cr)+fabs(ci) < FPMIN) cr=FPMIN;
                        den=dr*dr+di*di;
                        dr /= den;
                        di /= -den;
                        dlr=cr*dr-ci*di;
                        dli=cr*di+ci*dr;
                        temp=p*dlr-q*dli;
                        q=p*dli+q*dlr;
                        p=temp;
                        if (fabs(dlr-1.0)+fabs(dli) < EPS) break;
                }
                if (i > MAXIT) nrerror("cf2 failed in bessjy");
                gam=(p-f)/q;
                rjmu=sqrt(w/((p-f)*gam+q));
                rjmu=SIGN(rjmu,rjl);
                rymu=rjmu*gam;
                rymup=rymu*(p+q/gam);
                ry1=xmu*xi*rymu-rymup;
        }
        fact=rjmu/rjl;
        rj=rjl1*fact;
        rjp=rjp1*fact;
        for (i=1;i<=nl;i++) {
                rytemp=(xmu+i)*xi2*ry1-rymu;
                rymu=ry1;
                ry1=rytemp;
        }
        ry=rymu;
        ryp=xnu*xi*rymu-ry1;

        assignBessels(&tempBessels, rj, ry, rjp, ryp);

        return tempBessels;
}

void beschb(double x, double *gam1, double *gam2, double *gampl, double *gammi)
{
        double chebev(double a, double b, double c[], int m, double x);
        double xx;
        static double c1[] = {
                -1.142022680371172e0,6.516511267076e-3,
                3.08709017308e-4,-3.470626964e-6,6.943764e-9,
                3.6780e-11,-1.36e-13};
        static double c2[] = {
```

```
                1.843740587300906e0,-0.076852840844786e0,
                1.271927136655e-3,-4.971736704e-6,-3.3126120e-8,
                2.42310e-10,-1.70e-13,-1.0e-15};

        xx=8.0*x*x-1.0;
        *gam1=chebev(-1.0,1.0,c1,NUSE1,xx);
        *gam2=chebev(-1.0,1.0,c2,NUSE2,xx);
        *gampl= *gam2-x*(*gam1);
        *gammi= *gam2+x*(*gam1);
}

double chebev(double a, double b, double c[], int m, double x)
{
        void nrerror(char error_text[]);
        double d = 0.0, dd = 0.0, sv, y, y2;
        int j;

        if ((x-a)*(x-b) > 0.0) nrerror("x not in range in routine chebev");
        y2 = 2.0*(y=(2.0*x-a-b)/(b-a));
        for (j = m - 1; j >= 1; j--) {
                sv = d;
                d = y2*d - dd + c[j];
                dd = sv;
        }
        return y*d - dd + 0.5*c[0];
}

#undef EPS
#undef FPMIN
#undef MAXIT
#undef XMIN
#undef PI
#undef NRANSI

#undef NUSE1
#undef NUSE2
                                        #undef RTPIO2
```

APPENDIX J. ANSI C CODE FOR LEVIN ALGORITHM OPERATING ON EXTINCTION EFFICIENCY FACTOR

```
/* Thesis Program for applying the Levin method on the extinction efficiency factor and determining */
/* the resulting rate of convergence */
/* LT Brian Johnson */
/* Compiler:  Borland C++ Ver. 5.0 */
/* File Name:  mielevin.c */

#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <math.h>
#include "nrutil.h"
#include "nrutil.c"
#define NRANSI
#define EPS 1.0e-10
#define FPMIN 1.0e-30
#define MAXIT 10000
#define XMIN 2.0
#define PI 3.141592653589793
#define RTPIO2 1.2533141
#define NUSE1 5
#define NUSE2 5
#define MAX 1036

struct bessVec {
  double rj, ry, rjp, ryp;
};

struct complex {
  double real, imag;
};

void assignBessels(struct bessVec* besselsPtr, double rjdummy, double rydummy, double rjpdummy,
double rypdummy)
{
  besselsPtr -> rj = rjdummy;
  besselsPtr -> ry = rydummy;
  besselsPtr -> rjp = rjpdummy;
  besselsPtr -> ryp = rypdummy;
}

struct bessVec sphbes(double, int);          /* Function Prototype */
struct bessVec bessjy(double, double);       /* Function Prototype */

double levin();                              /* Function Prototype */
double bico(int, int);                       /* Function Prototype */
double factln(int);                          /* Function Prototype */
double gammln(double);                       /* Function Prototype */

struct bessVec sphBessels_n;
```

```
struct bessVec bessels_nPlusHalf;
struct bessVec bessels_nMinusHalf;
struct complex a_n, b_n;              /* Mie coefficients */
struct complex zeta_n;
struct complex zeta_nMinus1;
struct complex denom;                 /* denominator in Mie coefficients */

double x = 1000.0;                    /* size parameter */
double m = 2.0;                       /* refractive index (non-absorbing spheres) */
double J_nMinusHalf, J_nPlusHalf;
double A_factor;
double psi_nMinus1;                   /* Ricatti-Bessel fn 1st kind, initialized */
double psi_n;
double chi_n;                         /* Ricatti-Bessel fn 2nd kind */
double nOverx;
double aBraces, bBraces;
double numer;                         /* numerator in Mie coefficients */
double realDenom;                     /* rationalized denominator */
double sum_QextOld, sum_Qext, Q_ext;  /* Extinction efficiency factor */
double T[MAX+1], t[MAX+1];
double a;

int n = 0;                    /* loop counter, also order of bessel function */
int mArray[] = {1.25, 1.33, 1.44, 1.50, 1.55, 2.0};
int k;

main()
{
T[0] = t[0] = 0.0;
n = 0;
printf("\n\ni        T_i        t_i        P[i]/Q[i]\n");

bessels_nPlusHalf = bessjy(m*x, 0.5);      /* initialize for A_factor */
J_nPlusHalf = bessels_nPlusHalf.rj;

psi_n = sin(x);
zeta_n.real = psi_n;                       /* initializing zeta */
zeta_n.imag = cos(x);

sum_Qext =  0.0;

while (n<=MAX)
{
  n = n + 1;
  J_nMinusHalf = J_nPlusHalf;
  psi_nMinus1 = psi_n;
  zeta_nMinus1 = zeta_n;
  sum_QextOld = sum_Qext;

  bessels_nPlusHalf = bessjy(m*x, n + 0.5);      /* calculating A factor */
  J_nPlusHalf = bessels_nPlusHalf.rj;
  A_factor = (J_nMinusHalf/J_nPlusHalf) - (n/(m*x));
```

```
sphBessels_n = sphbes(x, n);        /* j's and y's for Ricatti Bess fn's */
psi_n = x * sphBessels_n.rj;
chi_n = -x * sphBessels_n.ry;
zeta_n.real = psi_n;
zeta_n.imag = chi_n;

nOverx = n/x;
aBraces = (A_factor / m + nOverx);
bBraces = (A_factor * m + nOverx);

numer = aBraces*psi_n - psi_nMinus1;

denom.real = (aBraces * zeta_n.real) - zeta_nMinus1.real;
denom.imag = (aBraces * zeta_n.imag) - zeta_nMinus1.imag;
realDenom = (denom.real*denom.real + denom.imag*denom.imag);

a_n.real = (numer * denom.real)/realDenom;
a_n.imag = (numer * denom.imag)/realDenom;

numer = bBraces*psi_n - psi_nMinus1;

denom.real = (bBraces * zeta_n.real) - zeta_nMinus1.real;
denom.imag = (bBraces * zeta_n.imag) - zeta_nMinus1.imag;
realDenom = (denom.real*denom.real + denom.imag*denom.imag);

b_n.real = (numer * denom.real)/realDenom;
b_n.imag = (numer * denom.imag)/realDenom;

t[n] =  2.0/(x*x)*(2*n + 1)*(a_n.real + b_n.real);
T[n] = T[n-1] + t[n];

}  /* end while loop */

a = levin();

printf("\nx = %3.2f  m = %1.2f \n\n", x, m);

return 0;
}

struct bessVec sphbes(double x, int n)
{
        struct bessVec tempsphbessels;
        struct bessVec bessels;

        double sj, sy, sjp, syp;
        void nrerror(char error_text[]);
        double factor, order;

        if (n < 0 || x <= 0.0) nrerror("bad arguments in sphbes");
        order=n+0.5;

        bessels = bessjy(x,order);
```

```
        factor=RTPIO2/sqrt(x);
        sj=factor*bessels.rj;
        sy=factor*bessels.ry;
        sjp=factor*bessels.rjp-(sj)/(2.0*x);
        syp=factor*bessels.ryp-(sy)/(2.0*x);

        assignBessels(&tempsphbessels, sj, sy, sjp, syp);

        return tempsphbessels;
}

struct bessVec bessjy(double x, double xnu)
{
        void nrerror(char error_text[]);

        struct bessVec tempBessels;

        double rj, ry, rjp, ryp;

        void beschb(double x, double *gam1, double *gam2, double *gampl,
                double *gammi);
        int i,isign,l,nl;
        double a,b,br,bi,c,cr,ci,d,del,del1,den,di,dlr,dli,dr,e,f,fact,fact2,
                fact3,ff,gam,gam1,gam2,gammi,gampl,h,p,pimu,pimu2,q,r,rjl,
                rjl1,rjmu,rjp1,rjpl,rjtemp,ry1,rymu,rymup,rytemp,sum,sum1,
                temp,w,x2,xi,xi2,xmu,xmu2;


        if (x <= 0.0 || xnu < 0.0) nrerror("bad arguments in bessjy");
        nl=(x < XMIN ? (int)(xnu+0.5) : IMAX(0,(int)(xnu-x+1.5)));
        xmu=xnu-nl;
        xmu2=xmu*xmu;
        xi=1.0/x;
        xi2=2.0*xi;
        w=xi2/PI;
        isign=1;
        h=xnu*xi;
        if (h < FPMIN) h=FPMIN;
        b=xi2*xnu;
        d=0.0;
        c=h;
        for (i=1;i<=MAXIT;i++) {
                b += xi2;
                d=b-d;
                if (fabs(d) < FPMIN) d=FPMIN;
                c=b-1.0/c;
                if (fabs(c) < FPMIN) c=FPMIN;
                d=1.0/d;
                del=c*d;
                h=del*h;
                if (d < 0.0) isign = -isign;
                if (fabs(del-1.0) < EPS) break;
        }
```

```
if (i > MAXIT) nrerror("x too large in bessjy; try asymptotic expansion");
rjl=isign*FPMIN;
rjpl=h*rjl;
rjl1=rjl;
rjp1=rjpl;
fact=xnu*xi;
for (l=nl;l>=1;l--) {
        rjtemp=fact*rjl+rjpl;
        fact -= xi;
        rjpl=fact*rjtemp-rjl;
        rjl=rjtemp;
}
if (rjl == 0.0) rjl=EPS;
f=rjpl/rjl;
if (x < XMIN) {
        x2=0.5*x;
        pimu=PI*xmu;
        fact = (fabs(pimu) < EPS ? 1.0 : pimu/sin(pimu));
        d = -log(x2);
        e=xmu*d;
        fact2 = (fabs(e) < EPS ? 1.0 : sinh(e)/e);
        beschb(xmu,&gam1,&gam2,&gampl,&gammi);
        ff=2.0/PI*fact*(gam1*cosh(e)+gam2*fact2*d);
        e=exp(e);
        p=e/(gampl*PI);
        q=1.0/(e*PI*gammi);
        pimu2=0.5*pimu;
        fact3 = (fabs(pimu2) < EPS ? 1.0 : sin(pimu2)/pimu2);
        r=PI*pimu2*fact3*fact3;
        c=1.0;
        d = -x2*x2;
        sum=ff+r*q;
        sum1=p;
        for (i=1;i<=MAXIT;i++) {
                ff=(i*ff+p+q)/(i*i-xmu2);
                c *= (d/i);
                p /= (i-xmu);
                q /= (i+xmu);
                del=c*(ff+r*q);
                sum += del;
                del1=c*p-i*del;
                sum1 += del1;
                if (fabs(del) < (1.0+fabs(sum))*EPS) break;
        }
        if (i > MAXIT) nrerror("bessy series failed to converge");
        rymu = -sum;
        ry1 = -sum1*xi2;
        rymup=xmu*xi*rymu-ry1;
        rjmu=w/(rymup-f*rymu);
} else {
        a=0.25-xmu2;
        p = -0.5*xi;
        q=1.0;
```

```
        br=2.0*x;
        bi=2.0;
        fact=a*xi/(p*p+q*q);
        cr=br+q*fact;
        ci=bi+p*fact;
        den=br*br+bi*bi;
        dr=br/den;
        di = -bi/den;
        dlr=cr*dr-ci*di;
        dli=cr*di+ci*dr;
        temp=p*dlr-q*dli;
        q=p*dli+q*dlr;
        p=temp;
        for (i=2;i<=MAXIT;i++) {
                a += 2*(i-1);
                bi += 2.0;
                dr=a*dr+br;
                di=a*di+bi;
                if (fabs(dr)+fabs(di) < FPMIN) dr=FPMIN;
                fact=a/(cr*cr+ci*ci);
                cr=br+cr*fact;
                ci=bi-ci*fact;
                if (fabs(cr)+fabs(ci) < FPMIN) cr=FPMIN;
                den=dr*dr+di*di;
                dr /= den;
                di /= -den;
                dlr=cr*dr-ci*di;
                dli=cr*di+ci*dr;
                temp=p*dlr-q*dli;
                q=p*dli+q*dlr;
                p=temp;
                if (fabs(dlr-1.0)+fabs(dli) < EPS) break;
        }
        if (i > MAXIT) nrerror("cf2 failed in bessjy");
        gam=(p-f)/q;
        rjmu=sqrt(w/((p-f)*gam+q));
        rjmu=SIGN(rjmu,rjl);
        rymu=rjmu*gam;
        rymup=rymu*(p+q/gam);
        ry1=xmu*xi*rymu-rymup;
}
fact=rjmu/rjl;
rj=rjl1*fact;
rjp=rjp1*fact;
for (i=1;i<=nl;i++) {
        rytemp=(xmu+i)*xi2*ry1-rymu;
        rymu=ry1;
        ry1=rytemp;
}
ry=rymu;
ryp=xnu*xi*rymu-ry1;

assignBessels(&tempBessels, rj, ry, rjp, ryp);
```

```
        return tempBessels;
}

void beschb(double x, double *gam1, double *gam2, double *gampl, double *gammi)
{
        double chebev(double a, double b, double c[], int m, double x);
        double xx;
        static double c1[] = {
                -1.142022680371172e0,6.516511267076e-3,
                3.08709017308e-4,-3.470626964e-6,6.943764e-9,
                3.6780e-11,-1.36e-13};
        static double c2[] = {
                1.843740587300906e0,-0.076852840844786e0,
                1.271927136655e-3,-4.971736704e-6,-3.3126120e-8,
                2.42310e-10,-1.70e-13,-1.0e-15};

        xx=8.0*x*x-1.0;
        *gam1=chebev(-1.0,1.0,c1,NUSE1,xx);
        *gam2=chebev(-1.0,1.0,c2,NUSE2,xx);
        *gampl= *gam2-x*(*gam1);
        *gammi= *gam2+x*(*gam1);
}

double chebev(double a, double b, double c[], int m, double x)
{
        void nrerror(char error_text[]);
        double d = 0.0, dd = 0.0, sv, y, y2;
        int j;

        if ((x-a)*(x-b) > 0.0) nrerror("x not in range in routine chebev");
        y2 = 2.0*(y=(2.0*x-a-b)/(b-a));
        for (j = m - 1; j >= 1; j--) {
                sv = d;
                d = y2*d - dd + c[j];
                dd = sv;
        }
        return y*d - dd + 0.5*c[0];
}

double levin()
{
 double P[MAX+1], Q[MAX+1];
 double commonTerm;
 int N;
 P[0] = Q[0] = 0.0;

 for (N = 1; N <= MAX; N++)
 {
  for (k = 1; k <= N; k++)
  {
   commonTerm = pow(-1,k)*pow(k,N-1)/t[k]*bico(N,k);
   P[k] = P[k-1] + commonTerm*T[k];
```

```
    Q[k] = Q[k-1] + commonTerm;
   }
   printf("\n%d        %f    %f     %f ", k - 1, T[N], t[N], P[N]/Q[N]);
  }
  printf("\n\n");
  return (P[MAX]/Q[MAX]);
}

double bico(int n, int k)
{       double factln(int n);
        return floor(0.5+exp(factln(n)-factln(k)-factln(n-k)));
}

double factln(int n)
{       double gammln(double xx);
        static double a[101];
        if (n <= 1) return 0.0;
        if (n <= 100) return a[n] ? a[n] : (a[n]=gammln(n+1.0));
        else return gammln(n+1.0);
}

double gammln(double xx)
{       double x,y,tmp,ser;
        static double cof[6]={76.18009172947146,-86.50532032941677,
                24.01409824083091,-1.231739572450155,
                0.1208650973866179e-2,-0.5395239384953e-5};
        int j;
        y=x=xx;
        tmp=x+5.5;
        tmp -= (x+0.5)*log(tmp);
        ser=1.000000000190015;
        for (j=0;j<=5;j++) ser += cof[j]/++y;
        return -tmp+log(2.5066282746310005*ser/x);
}

#undef EPS
#undef FPMIN
#undef MAXIT
#undef XMIN
#undef PI
#undef NRANSI

#undef NUSE1
#undef NUSE2
#undef RTPIO2
```

LIST OF REFERENCES

1. Illingworth, V., *Dictionary of Physics*, London: Penguin Books, Ltd., 1991.

2. Biblarz, O. and Netzer, D.W., *Evaluation of UTSI-CLA Program on Optical measurements of Turbine Engine Exhaust Particulates*, Monterey: Naval Postgraduate School, 1994.

3. Hansen, M.G. and Crum, L.A., *Mie Scattering as a Technique for the Sizing of Air Bubbles*, Mississippi: University of Mississippi, 1983.

4. Diermendjian, D., "Scattering and Polarization Properties of Water Clouds and Hazes in the Visible and Infrared", *Applied Optics*, Vol. 3, pp. 187-196, 1964.

5. Fahler, T.S. and Bryant, H.C., "Optical Back Scattering from Single Water Droplets", *American Journal Optics Society*, Vol. 58, pp. 304-310, 1968.

6. Chromey, F.C., "Evaluation of Mie Equations for Colored Spheres", J. Opt. Soc. Am., Vol. 50, pp. 730-737, 1960.

7. Stull, V.R. and Plass, G.N., "Emissivity of Dispersed Carbon Particles", J. Opt. Soc. Am., Vol. 50, pp. 121-129, 1960.

8. Lam, D.M. and Rossiter, B.W., "Chromoskedasic Painting", Scientific American, Nov. 1991.

9. Domb, C. and Lebowitz, J.L., *Phase Transitions and Critical Phenomena*, San Diego: Academic Press Inc., 1989.

10. Abramowitz, M. and Stegun, I.A., *Handbook of Mathematical Functions*, New York: Dover Publications, Inc., 1972.

11. Brezinski, C. and Zaglia, M.R., *Extrapolation Methods Theory and Practice*, New York: Selsevier Science Publishing Co., 1991.

12. Diermendjian, D., *Electromagnetic Scattering on Spherical Polydispersions*, New York: American Elsevier Publishing Company, 1969.

13. Edde, B., *Radar Principles, Technology, Applications*, New Jersey: Prentice Hall, 1993.

14. Van de Hulst, H.C., *Light Scattering by Small Particles*, New York: Dover Publications, Inc., 1981.

15. Wiscombe, W.J., "Mie Scattering Calculations: Advances in Technique and Fast, Vector-Speed Computer Codes", Boulder, Colorado: National Center for Atmospheric Research,1996.

16. Press, W.H., *Numerical Recipes in C, 2nd Edition*, New York: Cambridge University Press, 1992.

17. Levin, D., *International Journal of Computer Math*, Vol. 3, pg. 371, 1973.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center..... 2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, Virginia 22060-6218

2. Dudley Knox Library..... 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

3. Professor William Maier..... 1
   Chairman, Department of Physics, Code PH/Mw
   Naval Postgraduate School
   Monterey, California 93943

4. Professor James Luscombe..... 2
   Department of Physics, Code PH/Lj
   Naval Postgraduate School
   Monterey, California 93943

5. LT Brian E. Johnson..... 2
   11213 Wild Oak Dr.
   Oakdale, CA 95361

6. Marion Johnson..... 1
   7300 S. Forbes Rd.
   Lincoln, CA 95646

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHO
MONTEREY CA 93943-510

DUDLEY KNOX LIBRARY
3 2768 00342148 8